Slicing of Web Applications using Source Code Analysis

B.Tech Project Thesis

by

Ankit Kumar and Anubhav Panda
111CS0129 and 111CS0044

under the guidance of

Prof. D.P. Mohapatra

Department of Computer Science & Engineering
National Institute of Technology, Rourkela
Rourkela-769008, Odisha, INDIA
May, 2015

Department of Computer Science and Engineering
National Institute of Technology, Rourkela
Rourkela-769008, Odisha, India

May 11, 2015

Certificate

This is to certify that the work in the thesis entitled *Slicing of Web Applications using Source Code Analysis* by Ankit Kumar and Anubhav Panda is a record of an original research work carried out under my supervision and guidance in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in Computer Science and Engineering. Neither this thesis nor any part of it has been submitted for any degree or academic award elsewhere.

Durga Prasad Mohapatra
Associate Professor,
Dept of Computer Science and Engineering,
NIT Rourkela

Acknowledgement

First of all, we would like to express our deep sense of respect and gratitude towards our supervisor, Prof. Durga Prasad Mohapatra, who has been the guiding force behind this work. We want to thank him for introducing us to the field of Program Slicing and giving us the opportunity to work under him. His undivided faith in this topic and ability to bring out the best of analytical and practical skills in people has been invaluable in tough periods. Without his invaluable advice and assistance it would not have been possible for us to complete this thesis. We are greatly indebted to him for his constant encouragement and invaluable advice in every aspect of our academic life. We consider it our good fortune to have got an opportunity to work with such a wonderful person.

We would also like to thank all faculty members, PhD scholars and all colleagues to provide us their regular suggestions and encouragements during the whole work. We would like to specially thank Mr. Jagannath Singh for their constant words of enlightenment and wisdom, and their ideas and help whenever required.

At last but not the least we are in debt to our families to support us regularly during our hard times.

We wish to thank all faculty members and secretarial staff of the CSE Department for their sympathetic cooperation.

Ankit Kumar
Anubhav Panda

Abstract

Program slicing revealed a useful way to limit the search of software defects during debugging and to better understand the decomposition of the application into computations. The web application is very widely used for spreading business through out the world. To meet the desire of the customers, web applications should have more quality and robustness. Slicing, in the field of web application, helps disclosing relevant information and understanding the internal system structure. This in-terns help in debugging, testing and in improving the program comprehensibility.

The system dependence graph is an appropriate data structure for slice computation, in that it explicitly represents all dependencies that have to be taken into account in slice determination. We have extended the system dependence graph to Web-Application Dependence Graph (WADG). We have developed a partial tool for automatic generation of the WADG and computation of slices. In our literature survey, we found that most of the automatic graph generation tools are byte-code based. But, our tool uses the dependency analysis form the source code of the given program. We have presented three case studies by taking open source web programs and applying our techniques and slicing algorithm. We have found that the slices computed is correct and precise, which will be help full for program debugging and testing.

Construction of the system dependence graph for Web applications is complicated by the presence of dynamic code. In fact, a Web application builds the HTML code to be transmitted to the browser at run time. Knowledge of such code is essential for slicing.

Contents

Chapter 1

Introduction

During the early stages of website analysis, the main focus used to be on the evolution of entities present in the website. These entities where characterized by some defined metrices. Program Slicing is the static analysis of program statements used to decompose the program and extract statements from it pertaining to one specific operation or computation.program slicing is used in different fields such as debugging ,reverse engineering.It can be used in the field of software testing, program maintenance, reuse of programs, software safety, and metrics. A Web application consists of a set of Web pages displayed to the user and a set of server-side programs (usually scripts), performing some operations and resulting output pages to be displayed.

1.1 JSP Web Applications

Java Server Pages (JSP) web applications focuses more on presentation logic. They are easier to maintain than a servlet. As opposite to servlets, JSP adds Java code inside HTML rather than adding HTML code inside Java code but can still achieve everything that a servlet can do.

We will present an approach to obtain Web application slicing. A Web application is a special case of client-server system, in which the Web server or web services plays the role of the server,the Web browser loads the client side pages which plays the role of the client which establishes communication between the client and the server by using some specific protocols (HTTP, HTTPS). Due to the process of modularization most of the current applications use the service of web servers or web APIs (Application Programming Interfaces).

1.2 Need for Web Application Slicing

With the development of internet the quality of web applications has also increased and is continuing to do so. Slicing a web application with respect to a slicing criterion results in a simpler web application exhibiting similar behavior to the original web application. When web application slicing is used with other useful software, it proves to be very useful to programmers, developers, web designers, software engineers in numerous tasks such as debugging and regression testing. [1][2][3][4].

1.3 Proposed Work

In this thesis we have proposed our approach to handle the web pages which consists of JSP ,HTML and JavaScript parts. We have processed the 3 parts differently and generated the WADG (Web Application Dependence Graph) in order to explain the flow of web applications as well as different kind of dependence between them. We have considered page call and function call differently. Also, we have applied our slicing algorithm upon the generated WADG with multiple Slicing criteria and analyzed the resulting slices.We are also working on the dynamic slicing of web application. And we have considered different cases of it.

1.4 Problem of dynamic code generation

We often come across web applications that contain web pages generated dynamically. Such applications are far more difficult to analyze compared to static websites. HTML construct is built during runtime as server scripts in the pages are executed. Each and every thing starting from names of variables in the resulting page to the FORM and its ACTIONS and even the HYPERLINKS are generated dynamically during runtime, and thus can change for every execution during runtime, depending on program state or input values from the user or database.

For instance, say we ignore the generated code while construction of WADG, it will result in the absence of several nodes and edges from the WADG. For example, WADG built for server scripts do not have nodes for dynamically generated tags, and even misses data dependencies for dynamic variable names and call dependencies for dynamic FORM ACTIONS. It is almost impossible to construct a WADG with great precision for web applications containing dynamic code by source code analysis only as the nodes of the WADG which are generated dynamically will be very difficult to approximate statically. But, it will be safe to assume that static analysis will be able to construct WADG with great precision for a subset of all web applications possible that will include the most important real world applications.

We can also construct dynamic portions of WADG by a separate method i.e, dynamic slicing. We can run the web application itself and consider a particular execution of it with a given set of inputs. This completely eliminates the problem of approximation of generated HTML markup. This way, slices can be easily obtained from the known generated markup. The only disadvantage of this approach is that the slices computed will only be true for those specific inputs. Hence, we will use static slicing.

1.5 Motivation

Software architecture is an emerging area in software development. Whenever we deal with large scale software intensive system software architecture documentation becomes essential which helps throughout development of the software system. In this section we have cover an overview of the area of investigation, the motivation for this work.

- Web applications are becoming huge day by day.

- Static web applications are used widely in firms which needs debugging techniques.
- Most web applications are dynamic in nature, debugging them becomes even more important.
- Slicing of Web Applications helps a great deal with applications like debugging, code understanding, software testing, reverse engineering, program maintenance, reuse, software safety and metrics.

1.6 Objectives

The objectives of the work are fixed as follows:

- To study the importance of web slicing.
- To generate SDG for web applications.
- To propose WADG for web applications.
- To implement static slicing in java server pages.
- To implement dynamic slicing in java server pages.

1.7 Organization of Thesis

Chapter 1 discuses a brief introduction to JSP web applications and their types. It also emphasizes on the need for slicing of web applications and its widespread advantages. It also further informs about the problem of dynamic code generation and the amount of difficulty it brings to the process of slicing. Then it describes the objectives and motivation behind our work. Chapter 2 discuses all the basic concepts and terminologies that has been used in the paper to describe the work done. Slicing and its types, SDG and various dependencies along with the special type of dependence graph for web applications i.e. the WADG along with the new additional dependencies were discussed at length in this chapter. Chapter 3 gives the literature review of our work that i.e. it gives a brief account of the related work by various authors. Chapter 4 discuses in detail about the procedure and algorithms followed for generation of the web application dependence graph. We also discussed the techniques and algorithms for code extrusion, string-cat propagation and flow information which are used to handle the presence of dynamic code in the web application. Small examples for each with corresponding WADGs were shown.We finally tested the approach with an example web application and generated its WADG. Chapter 5 discuses the slicing algorithm and its working is explained in detail along with the example web application and its WADG generated in the previous chapter. Furthermore, the utility of the work was shown with a comparison study with works of different authors. Chapter 6 discuses the overall implementation of the work by first describing the experimental setup used for performing various case studies. A tabular representation of the case studies along with their finding is shown. The overall model of the setup is shown with the help of a flow diagram. The flow diagram shows the various modules of WADG generation and slicing. Chapter 7 concludes the thesis and gives a brief account of the future work that can be continued from where we have left.

Chapter 2

Basic Concepts

A program slice is a reduced, executable program obtained from a given program by removing statements, so that it replicates part of the behavior of the initial program[5][6].
Similar concept has been applied on web applications to generate web application slicing. This can be achieved with the help of system dependence graph.

2.1 Slicing

A web application is made up of web pages and server side scripts which are compiled by web server and displayed by web browser. A user views a web application on a web browser, consisting of only HTML markup as all the scripts are compiled and converted to HTML.Thus slicing a web application with respect to a slicing criterion results in a simpler web application exhibiting similar behavior to the original web application. A computed web application slice consists of some HTML markup and scripts from the initial web application which are selected in order to replicate a small part or behavior of initial application.[7][8].

2.1.1 Slicing Criterion

The criterion for slice computation is an information item of interest displayed in a given Web page, and the resulting slice reproduces the same information item, if the user performs the same navigation actions and provides the same input in the original and in the sliced Web application. Formally, a slicing criterion (n, x) consists of a statement n, displaying the information item of interest, and a variable x, used for the production of such an information item[7].

2.1.2 Types of Slicing

There are many types of program slicing exists. Slicing can be distinguished by the nature of program's execution or by its approach. We are here presenting some basic types of slicing, but for more detailed study some good review papers [9][10], can be referred.

Static Slicing: A static slice includes all the statements that affect variable v for a set of all possible inputs at the point of interest (i.e., at the statement x). Static slices are computed by finding consecutive sets of indirectly relevant statements, according to data and control dependencies.

Dynamic Slicing: A dynamic slice contains all statements that actually affect the value of a variable at a program point for a particular execution of the program rather than all statements that may have affected the value of a variable at a program point for any arbitrary execution of the program.

Forward Slicing: Forward slice consists of all statements and control predicates dependent on the slicing criterion, a statement being "dependent" on the slicing criterion or information of interest if the values computed at that statement is responsible for the values computed at the slicing criterion, or if the values computed at the slicing criterion determine the fact if the current statement is executed or not.

Backward Slicing: A backward slice contains statements of a program which has some effect on the slicing criterion. It helps the developer to locate the parts of the program that contains a bug. Backward slice requires tracing dependencies in the backward direction.

2.2 System Dependence Graph

A System Dependence Graph (SDG) is used to model dependence between statements within a procedure including inter-procedural dependence. The final slicing algorithm will be depend over the SDG. The SDG is a graph whose nodes roughly correspond to program statements and whose edges model dependence in the program.

There are various types of dependence which could explain the different relation between two statements of the input program.

2.2.1 Control Dependence

A control dependence or nested dependence exists between two statements if one statement lies within the scope of the other statement For conditional and loop constructs, control dependence holds between the condition and the executable statements enclosed within those constructs as their execution depends on the truth value of the condition. A similar approach is followed for server side scripts, however for HTML markups, a control dependence also exists between an HTML tag and the statements enclosed within the tags.[7].

2.2.2 Data Dependence

A data dependence can exist between a pair of server side statements or between a HTML and server side statement if one statement uses one or more values of variables defined by other statement and a definition clear path i.e, a path without redefinition of variables, exists between the two statements. In most programs a data dependence usually exists between a statement which defines the value of a variable and another statement which uses that variable for calculation or display along a definition clear path.[7].

2.2.3 Call Dependence

A call dependence exists between an input tag with type submit, i.e, a submit button and the dynamic page specified in the action attribute of the tag. In other words, a call dependence exists between two pages if clicking on a button on one page redirects to the other page. When

a server side program is invoked from an HTML statement (e.g., via form submission or HREF), control of execution is transferred to the server and never returns back to the Web page issuing the call. In other words, a server program invoked from an HTML statement cannot produce any effect on the variables of the calling page, since the invocation is a no-return invocation. A consequence is that the calling context problem, i.e., the problem of keeping the data flows associated to the different call sites separated, is absent, since call sites are not affected by the invocation[7].

2.2.4 Parameter - In

A parameter-in dependence exists between an HTML input tag and server side request-form statement if request statement receives the input variable from the HTML input tag when called by the form containing the input tag.[7].

2.2.5 Parameter - Out

A parameter-out dependence holds between any value returned from an invocation and the related variable in the call site.

2.2.6 Summary Edge

In the case of slicing, summary edges represent a transitive data dependence through a particular call site. Summary edges connect actual-in vertices with actual-out vertices and represent a that dependence between the connected vertices may be created in the called procedure/pages or (transitively) through other called procedures/pages.

2.3 Web Application Dependence Graph

A System Dependence Graph is incapable of handling web applications and is only suitable for programs. Hence we introduce a Web Application Dependence Graph(WADG) with following additional dependences:-

2.3.1 Event Loop

Mouse or keyboard events occurring during or after page loading can trigger the execution of a client side procedure, can produce the loading of another page (hyperlinks), or the calling of a JavaScript. The graphical user interface of the browser handles such events by means of a so called event loop.

2.3.2 PageCall Dependence

A PageCall dependence holds between each HTML input statement of type submit and the dynamic page specified in the associated action.

```
aa.jsp
1 <%
2 z = 'aa';
3 %>
4 <FORM method="POST" action="bb.jsp">
5 <INPUT TYPE="Submit"><BR>
6 <INPUT TYPE="Text" NAME="x" VALUE=z><BR>
7 <INPUT TYPE="Text" NAME="y" VALUE=z><BR>
8 </FORM>
bb.jsp:
9 <%
10 var x', y';
11 x' = request.getParameter("x");
12 y' = request.getParameter("y");
13 %>
```

Figure 2.1: Sample JSP program

2.3.3 Build Dependence

A build dependence holds between a server program statement and an HTML statement if the former generates the latter. During slice computation, build dependences are traversed backward similarly to the other dependences. In this way, the statements in the original program responsible for generating the HTML statements included in the slice will be also part of the slice[11].

In the above example (in Figure-2.1), the first jsp page consists of two subsections of code, i.e, HTML and Java beginning at statements 1 and 4 respectively. Thus, there is a control dependence between the page name and the two statements. Similarly, there is a control dependence between statement 1 and all other Java statements, i.e 1 and also between statement 4 and all other HTML statements, i.e,5,6,7 enclosed within tag represented by statement 4.

Now, statements 6 and 7 get their value of x from statement 2, resulting in data dependence between statement 2 to 6 and 2 to 7.

Also, the page bb.jsp is called when the FORM in aa.jsp is submitted from click event of statement 5, resulting in a call dependence between statement 5 and bb.jsp. Values are passed from statement 6 to 11 and statement 7 to 12 resulting in a parameter-in dependence between those statements. The final SDG for the above example program is shown in Figure-2.2.

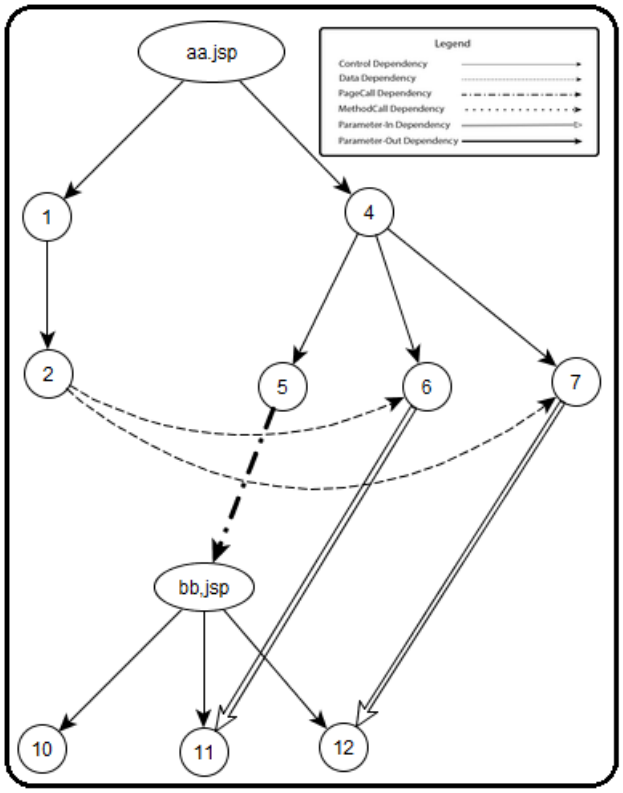

Figure 2.2: SDG of the program in Figure-2.1

2.4 Summary

Here we discussed all the basic concepts and terminologies that has been used in the paper to describe the work done. Slicing and its types, SDG and various dependencies along with the special type of dependence graph for web applications i.e, the WADG along with the new additional dependencies were discussed at length in this chapter.

Chapter 3

Related Works

In this section, we present a brief survey of the existing literatures those are closely related to our work.

3.1 Program Slicing

Binkley and Gallagher [9] used the concept of Procedural Dependence Graphs(PDGs) and used dependencies between them to construct System Dependence Graphs(SDGs) for programs. He then used SDGs for static and dynamic slicing as various problems such as Graph-Reach-ability problem, Data-Flow problem etc.

Weiser [5] has introduced program slicing and presented a data flow algorithm for approximating slices. He has investigated slicing empirically and highlighted importance of slicing in debugging applications.

Horwitz et al. [12] has worked on inter-procedural slicing using Procedural Dependence Graphs(PDGs) and System Dependence Graphs(SDGs). He has used concepts of attribute grammar, graph dependencies, flow analysis, flow-insensitive summary information and generated graphs and calculated slices using inter-procedural slicing in the presence of procedural calls, parameter passing and aliasing.

3.2 Slicing of Web Applications

A web application consists of, a set of web pages displayed to the user and a set of server side programs (usually scripts). The web application performs some computation and produces output pages to be displayed. A web application slice is obtained from a given set of web pages and scripts, by removing HTML and script statements, so that part of the behavior of the initial web application is replicated.

Ricca et al. [13] have used ReWeb tool for creating the UML diagram for web application. He has developed SDG for web-based programs taking different edges for control dependency, data dependency, call dependency and semantic dependency. He has implemented his slicing technique on a Travel Agency web application, developed using ASP and HTML.

Sahu et al. [14] have proposed an algorithm for slicing of JSP web applications. They have constructed the SDG for the JSP program. The slice is coputed by traversing backward in the graph and marking the edges and nodes.

In a paper by Junhua et al. [15], they have proposed a new method named Program Dependency Hyper Graph (PDHG) to describe the dependency of a web application using hyper graph theory. Also proposed an algorithm for slicing using PDHG.

Maras et al. [16], has proposed slicing of only client side web applications without considering server side dynamic code. They have built a firefox addon called firecrow on top of firebug which calculates the slice of static web applications. They have tested their slicing algorithm on open source math libraries.

Chapter 4

WADG For Web Application

When a JSP page is called from an HTML statement of another JSP page, (e.g. via FORM submission or HREF), control of execution is transferred to the newpage and never returns back to the web page issuing the call. In other words, a server program invoked from an HTML statement cannot produce any effect on the variables of the calling page, since the invocation is a no-return invocation.

Definition- *PageCall Dependence*: A PageCall dependence holds between each HTML input statement of type submit and the dynamic page specified in the associated action.

4.1 WADG Generation

Algorithm-1 is used to generate the WADG of a given JSP program. It basically has 3 sections - JavaScript, HTML and JSP. First it reads file by file from a given folder. Then it processes its JavaScript part. It constructs node, inserts control and data dependence between these nodes. It also stores function names to a HashMap, FuncMap. It also stores variable names to a HashMap, JSmap.

First the algorithm finds whether the code is in Html or not.If it is in Html it calls the algorithm htmlparser.In the HTML section it searches for <Form> tag and stores it's action in a HashMap, PageMap. It searches for <input> tags and stores their names in a HashMap, HTMLmap. If <input> tag contains a JavaScript function call, it adds a call dependency between this and matching nodes of FuncMap.ii.if in js function call it finds that html parser contains that parameters that have been passed to js function then it adds a summary edge between this and all entries of HTMLmap.This part is modularized because the Dynamic part also calls it. If that function contains parameter(s) then it adds param-in dependence between nodes from HTMLmap and JSMap. It process the JSP part at last. During this process, it constructs nodes for individual statements and inserts control and data dependencies between nodes. If line contains request. getparameter or any method through which it can access a parameter sent from other page, it adds param-in dependency between calling form node and current node.If it finds that a page returns some parameter from the page from which it was called it Adds param-out dependency between current node and destination form node.In order to handle the dynamic Web pages we have defined three algorithms.Since the Jsp part contains the dynamic code so the algorithm calls three different algorithm to handle that dynamic part.

After processing all the pages from a folder it adds PageCall dependency from each entry of PageMap to the corresponding pages.

Algorithm 1 WADG Generation

INPUT: P- Input program, I- Input set for P,
Slicing criterion s

OUTPUT: The Slice S for s.

```
1: for each JSP page do
2:     while !End of File do
3:         Line = readline
4:         if Line = comment then
5:             continue
6:         else if Line = JavaScript then
7:             a) handle the java script components by adding control dependence , data dependence and store the name of functions in a data structure.
8:         else if Line = HTML then
9:             a) htmlParse(line)
10:        else if Line = JSP then
11:            handle control, data dependence in it.
12:            handle param-in dependency,param-out dependency
13:            Add Summary edge between nodes.
14:            f) codeExtrusion(line)
15:            g) stringcatPropagation(line)
16:            h) flowInformation(line)
17: Add PageCall dependency
```

Algorithm 2 Html Parser

```
1: if < form > tag found then
2:     Store action name.
3: if < input > tag found then
4:     Store names
5:     if < input > contains jS function call then
6:         Add a call dependency
7: Add param-in dependency
```

We have taken a JSP program, as shown in Figure-4.1 and Figure-4.2 . This program takes an input from the user and calculate the factorial of the number and display it to the user. We have applied the Algorithm-1 and generated the WADG for the whole JSP program and shown in Figure-4.3.

4.2 WADG for dynamic code

The problem of statically determining the HTML code generated dynamically by a Web application is in general undecidable. Consequently, it is in general impossible to build an accurate SDG for a Web application that generates some HTML code at run time. Since it is not possible to determine the HTML code generated by a dynamic scripts in the general case, the typical patterns of code generation are considered and a technique to handle them is presented.

```
index.jsp
4  <html>
5  <head>
6  <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
7  <title>Insert title here</title>
8  <script>
9  function validateForm() {
10     var x = document.forms["form1"]["x"].value;
11   if(parseInt(x)!=x||parseInt(y)!=y || parseInt(q)!=q)
12         {
13                   alert("Enter Integers only as Input");
14                   return false;
15         }
16     else
17         return true;
18 }
19 </script>
20 </head>
21 <body>
22 <form action="factorial.jsp" method="post" name="form1" >
23 <input type="text" name="x" /><br>
24 <input type="submit" name="max" onclick="return validateForm()"/>
25 </form>
26 </br>
27 <%
28 String z;
29 z=request.getParameter("result");
30 if(z!=null)
31         out.println("<input type=\"text\" name=\"result\" value=\""+z+"\"/>");
32 %>
33 </body>
34 </html>
```

Figure 4.1: Example JSP program

```
factorial.jsp
4  <html>
5  <head>
6  <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
7  <title>Insert title here</title>
8  </head>
9  <body>
10 <%
11 String x,y,q;
12 int z,i;
13 x=request.getParameter("x");
14 z=Integer.parseInt(x);
15 for(i=z-1;i>1;i--)
16         z=z*i;
17 response.sendRedirect("index.jsp?result="+z);
18 %>
19 </body>
20 </html>
```

Figure 4.2: JSP program for calculating factorial of a number

4.2.1 Code Extrusion

A JSP page may contain print statements which prints HTML code. In such cases, the output of such HTML code can only be know at runtime making it impossible to generate the WADG with simple source code analysis. To deal with this, we propose an algorithm of code extrusion, which converts and replaces the print statements with their HTML output in the source code. Example Program and corresponding WADG are shown in Figure-4.4 and Figure-4.5

Algorithm 3 CodeExtrusion

1: **If** line contains System.out.println:
2: a. Replace the print statement with an unquoted version of the printed string
3: b. handle html statements.
4: c. Add build dependency between print statement and generated HTML statement.

4.2.2 String-Cat Propagation

A JSP page may contain variables initialized and concatenated with values which result in those variables containing HTML code. When print statements print such variables the same problem of dynamic code generation occurs as discussed in previous section, but cannot be resolved by code extrusion alone. To deal with this, we propose an algorithm of string-cat propagation, which converts and replaces the print statements with the values of variables, containing HTML code, in the source code. Example Program and corresponding WADG are shown in Figure-4.6 and Figure-4.7

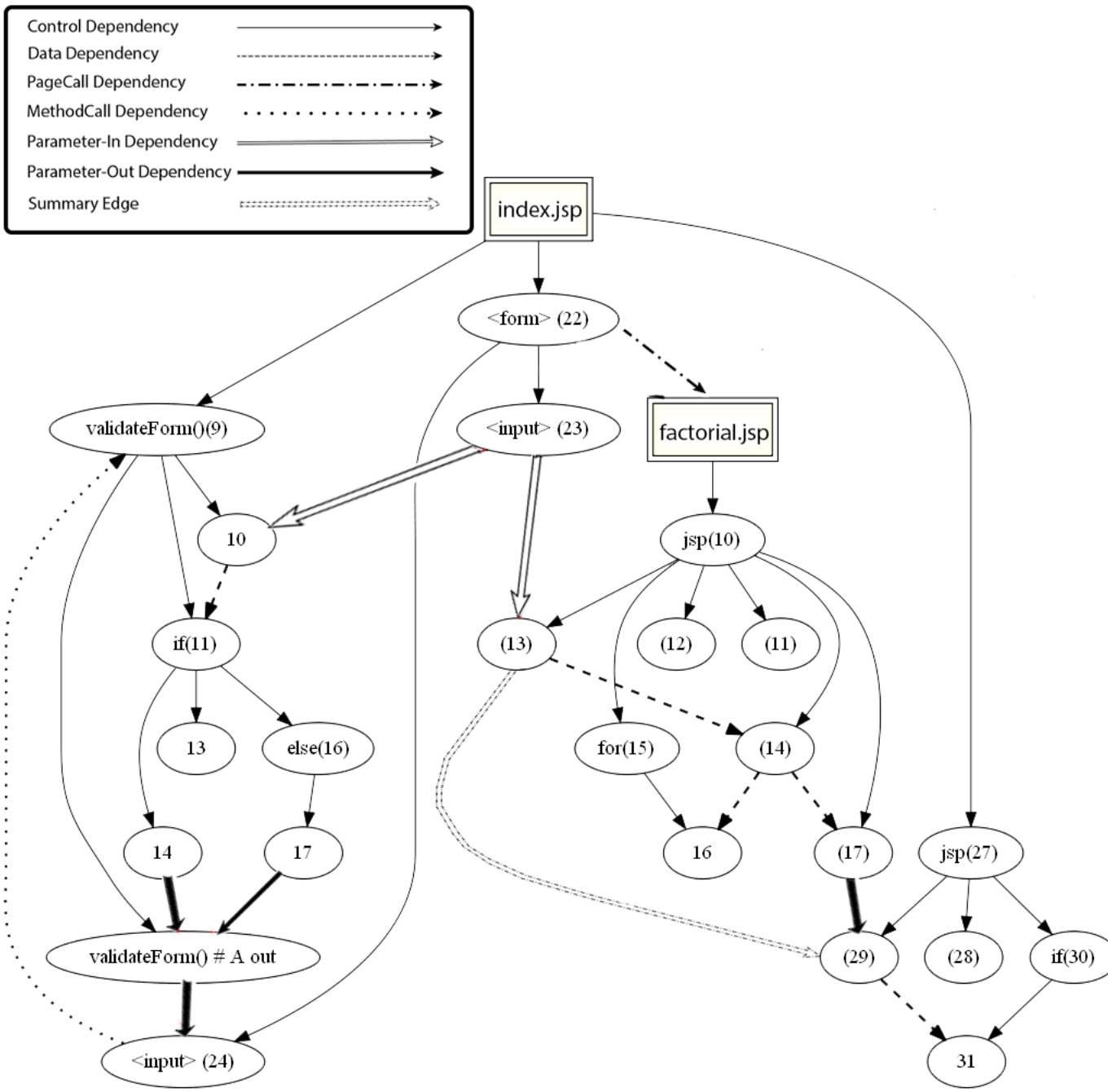

Figure 4.3: Generated WADG for the example programs in Figure-4.1 and Figure-4.2

```
10 <%
11 out.println("<form action=\"factorial.jsp\" method=\"post\" name=\"form1\" >");
12 out.println("<input type=\"text\" name=\"x\" /><br>");
13 out.println("<input type=\"submit\" name=\"max\"/>");
14 out.println("</form>");
15 %>
```

Figure 4.4: Code Extrusion Example

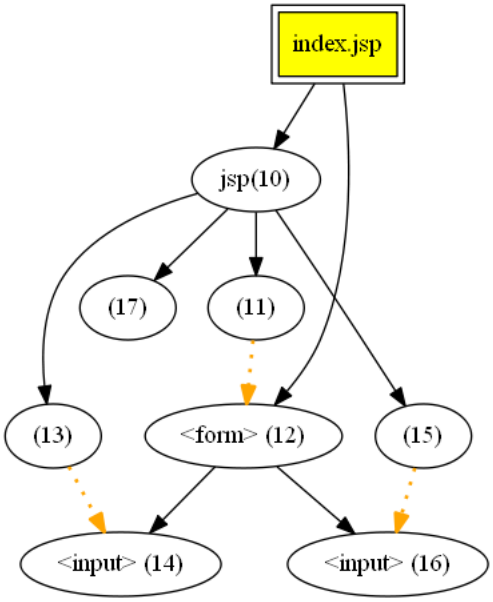

Figure 4.5: Code Extrusion Example Graph

Algorithm 4 StringCat Propagation

1: **if** line is a string assignment:
2: a store it.
3: **else if** line is a string concatenation :
4: a. update it
5: **else if** line contains System.out.print: i. codeExtrusion(line).

```
10 <%
11 String st;
12 st="<form action=\"factorial.jsp\" method=\"post\" name=\"form1\" >";
13 st=st+"<input type=\"text\" name=\"x\" /><br>";
14 st=st+"<input type=\"submit\" name=\"max\"/>";
15 st=st+"</form>";
16 out.println(st);
17 %>
```

Figure 4.6: String Cat Propagation Example

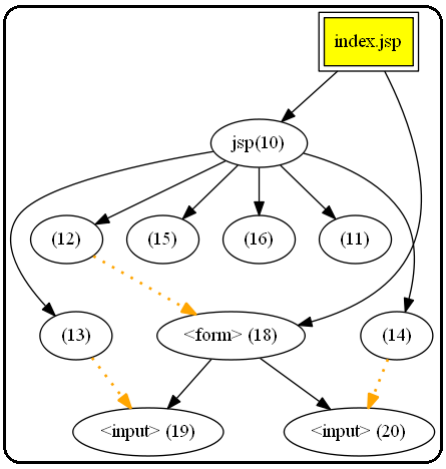

Figure 4.7: String Cat Propagation Example Graph

4.2.3 Flow Information

It might happen that a given JSP variable is associated to more than one string-cat after flow propagation. This occurs, for example, when alternatives are in the code(such as If Else construct). To deal with this, we propose an algorithm of flow information , which converts and replaces the print statements with an <ALT><CASE> HTML construct. In the presence of loops,an <ALT><CASE> construct with k(properly set) iterations is considered. Example Program and corresponding WADG are shown in Figure-4.8 and Figure-4.9

Algorithm 5 FlowInformation

1: **if** line conatins if or line conatins else:
2: a. handle it by introducing alt and case tags.
3: **else if** line conatains loop:
4: a. handle it by using alt case tags and using a parameter k i.e. no of iterations.
5: **else if** line contains System.out.print:
6: i. codeExtrusion(line).

4.3 Summary

In this chapter we discussed in detail about the procedure and algorithms followed for generation of the web application dependence graph. We also discussed the techniques and algorithms for code extrusion, string-cat propagation and flow information which are used to handle the presence of dynamic code in the web application. Small examples for each with corresponding WADGs were shown.We finally tested the approach with an example web application and generated its WADG.

```
10 <form action="a.jsp" method="post" name="form1" >;
11 <%
12        String st="";
13        int a=1;
14        if(a==1)
15        {
16                st=st+"<input name=\"x\">";
17        }
18        else
19        {
20                st=st+"<input name=\"y\">";
21        }

22        out.println(st);
23 %>
24 </form>
```

Figure 4.8: Flow Information Example

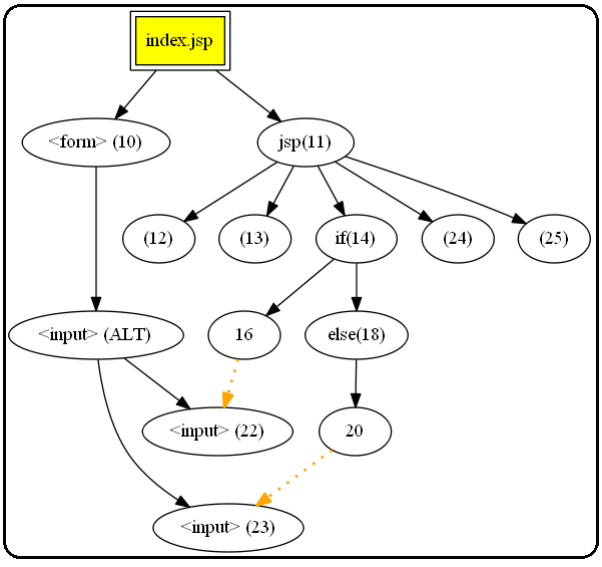

Figure 4.9: Flow Information Example Graph

Chapter 5

Slicing of Web Applications using WADG

5.1 Slicing Algorithm

Algorithm 6 Slicing Algorithm : Two Phase

INPUT: G- A WADG, s- Slicing Criterion,

OUTPUT: The Slice S for s.
lists and the result set

```
1:      W1 ={s}, W2 = {}, S ={s} //two work lists and the result set
       /* phase 1 */
2: repeat
3:      W1 =W1/ {n} // process the next node in W1
4:      for all m− >e n // handle all incoming edges of n
5:          if m ∉ S // m has not been visited yet
6:              S = S∪{m} // if e is not a param-out edge, add m to W1, otherwise, add
   m to W2
7:                  if e ∉ {po}
8:                      W1 = W1 ∪ {m}
9:                  else
10:                     W2 = W2 ∪ {m}
11:     Until W1 = ∅
       /* phase 2 */
12: repeat
13:     W2 =W2/ {n} // process the next node in W2
14:     for all m− >e n // handle all incoming edges of n
15:         if e ∉ {pi, call}
16:                 W2 = W2 ∪ {m}
17:                 S = S ∪ {m}
18:     Until W2 = ∅
19:     return S
20:
```

After the construction of WADG for a given JSP program, we have to compute slices by taking different slicing criteria. We have used a two-phase slicing algorithm [12], as shown in Algorithm-2, for our slice computation. Slicing as described in previous section is the process

of selecting statements from the given web application with respect to an information of interest(slicing criterion) that replicates a part of the application itself. Sticking to this definition, Algorithm 2 takes an WADG of the web application and a slicing criterion as input, and gives a selected set of nodes from the WADG called slice as output. It performs the slicing operation in 2 phases, hence the name. The Algorithm recursively marks nodes of the WADG, starting with the slicing criterion itself, then proceeding to all its incoming edges.

In the first phase, all edges except parameter-out edges are considered for marking.In the second phase, the parameter-out edges left in phase one are considered separately. This is done in order to avoid some undesirable nodes from being marked, as they are unrelated to the slicing criterion and should not be marked. As seen from the phase two of the algorithm, all the parameter-in and call edges are ignored and not added to slice.Think of this like a portion of code in one page gives an output which gets received and stored at some statement(say ST) of some other page. Now, in this situation, that particular portion of code has absolutely no relation with any other code which calls ST or gives an output to ST, hence should not be included in the slice, as achieved by the second phase of the algorithm.

To explain the working of the slicing algorithm, we have used the same example WADG shown in Figure-4.3. Suppose the slicing criterion for this example is $s = 31$. By using the slicing algorithm we have computed slice and shown it in Figure-5.1. Here in this figure, the nodes included in the slice is shown as shaded nodes.

5.2 Comparison Study

A web application consists of set of web pages displayed to the user and a set of server side programs (usually scripts). The web application performs computation and produces output pages to be displayed.

A web application slice is obtained from a given set of web pages and scripts by removing HTML and CSS statements, so that part of the behavior of the initial web application is replicated.

In the paper, Web Application Slicing(2001), they have considered simple static web application, without haveing dynamic code. They have not considered JavaScript as well as event loops. They have used simple backward traversal for slicing and that does not give precise results, i.e. the slice contain more nodes than the expected result.

In the paper Slicng Java Server Pages application(2008) by M. Sahu and D.P. Mohapatra, they have considered static pages without having dunamic code. They have not considered JavaScript and event loop. They have constructed the SDG using procedure dependent graph and for some specific HTML tags only. They have used simple backward traversal algorithm for sliced output.

In the paper, Construction of Web Application Slicing in the presence of Dynamic Code Generation by Ricca and Tonella, they have proposed different approaches to construct SDG for dynamic codes, but have not proposed an algorithm for it. They have not discussed slicing methods and not proposed any algorithm for it.

In the paper, Slicing Web Applications based on hyper graph, they have considered the generation of hyper graphs by a new method. They have used Program Dependency hyper graph

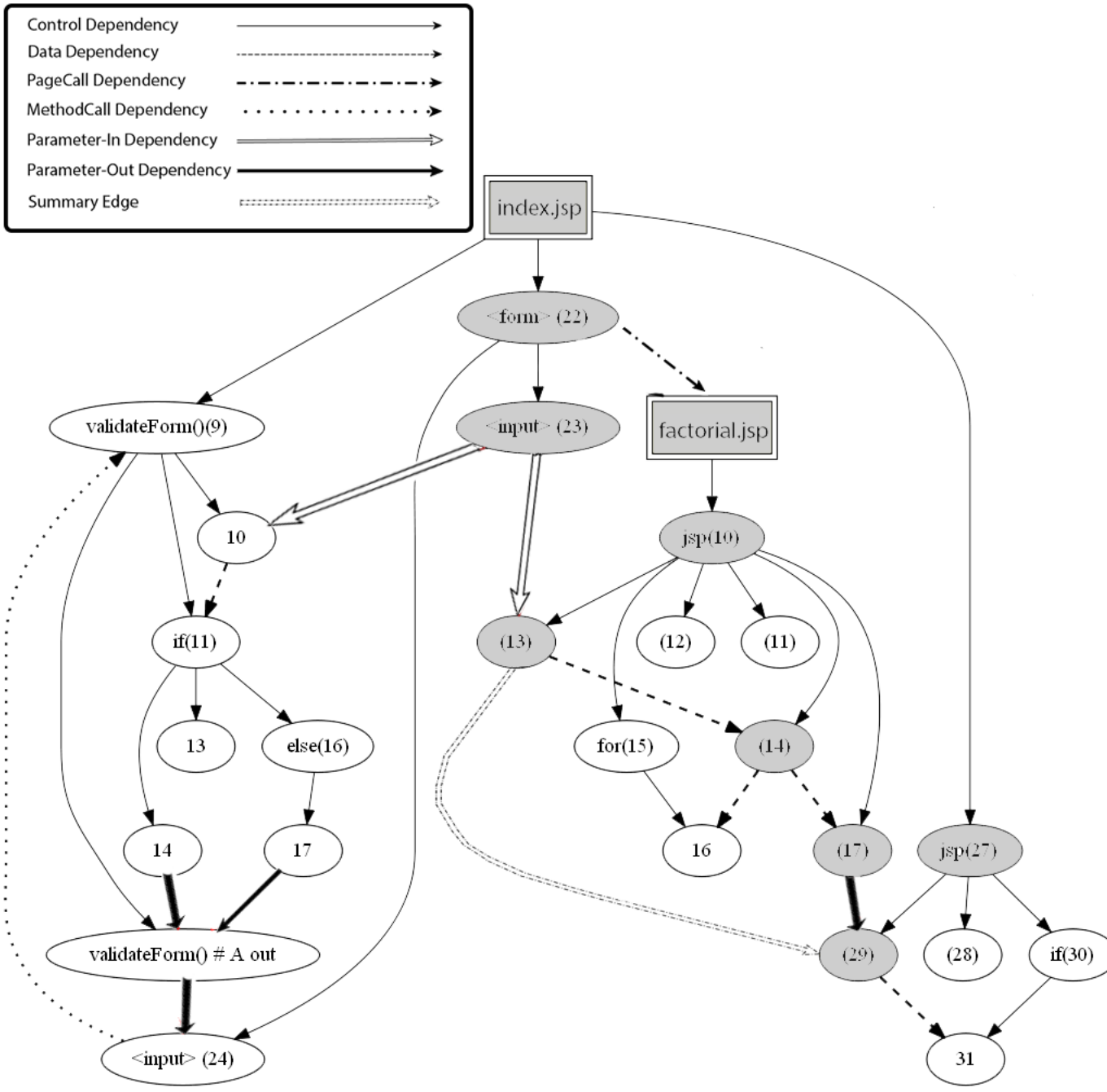

Figure 5.1: Generated slice for WADG in Figure-4.3 w.r.t. $s = 31$

to describe dependency of web application using hyper graph theory. They have also proposed an algorithm for slicing using PDHG. They have not considered JavaScript and dymanic code etc.

In the paper, Client Side Web Application Slicing by Maras, they have proposed slicing of frontend web applications. They have considered the HTML and JavaScript part but not the JSP part. In other words, they have not considered the static and dynamic aspect of web pages.

Table 5.1: Outcome of the comparative studies

Paper Name	**Static Slicing**	**Dynamic Slicing**	**JavaScript**	**Summary Edge**	**Two Phase Algorithm**
Ricca 2001	Yes	NO	NO	NO	NO
Ricca 2005	Yes	Yes	NO	NO	NO
Sahu 2008	Yes	NO	NO	NO	NO
Junhua 2009	Yes	NO	NO	NO	NO
Maras 2011	Yes	NO	NO	NO	NO
Our Paper	Yes	Yes	Yes	Yes	Yes

5.3 Summary

In this chapter, slicing algorithm was presented and its working was explained in detail along with the example web application and its WADG generated in the previous chapter. Furthermore, the utility of the work was shown with a comparison study with works of different authors.

Chapter 6

Implementation and Discussion

We have developed a partial tool for automatic generation of WADG for a given JSP program. In the following section we are presenting the detailled implementation of our tool. Later, we have taken some real case studies to validate the working of our developed tool and slicing algorithm. We have performed the case studies with a personal computer having Intel Core $i5$ processor, clock speed 2.40GHz, primary memory 4 GB and Windows 7 Home Basic (64 bit) operating system.

6.1 Experimental Setup

In the block diagram, shown in Figure-6.1, there are six sections that reads and analyzes each and every file of the web application sequentially and generates the WADG.
At first the file is processed by JavaScript Analyzer that finds all the function names and variables names and stores them in different hash maps in key value pair. It also computes control and data dependence between JavaScript statements and sends results to the combined output section.

In the next section, HTML analyzer looks for two tags, i.e.$< input >$ and $< form >$ tags. If they contain any JavaScript function calls, a call dependency is added between them and matching nodes of the function map generated by the JavaScript section. If that function contains parameters, then a param-in dependence is added between nodes from HTMLmap and JSMap. Moreover, it computes control dependence between nodes and sends the results to the combined output section.

Finally, JSP analyzer process the JSP part. Control and data dependence between nodes is computed followed by a check for param-in dependence by checking the methods. Furthermore, it also checks if a page returns any parameter from which it was called and adds param-out dependence. If the JSP analyzer detects presence of dynamic code, it sends the dynamic code for further processing to a three step dynamic code processor made of Flow Information, String Cat Propagation and Code Extrusion. The working of each has been explained in section 5.2. The total processed result is sent to the combined output section of 3 analyzers. In the combined output of 3 sections, page call dependence is added between pages by checking the entry of the hash map provided by HTML section. Then it gives its output to the WADG generator section.

The WADG generator section receives the output of the previous sections and generates the WADG by writing a .gv file in the system. It adds all the dependencies according to the .gv

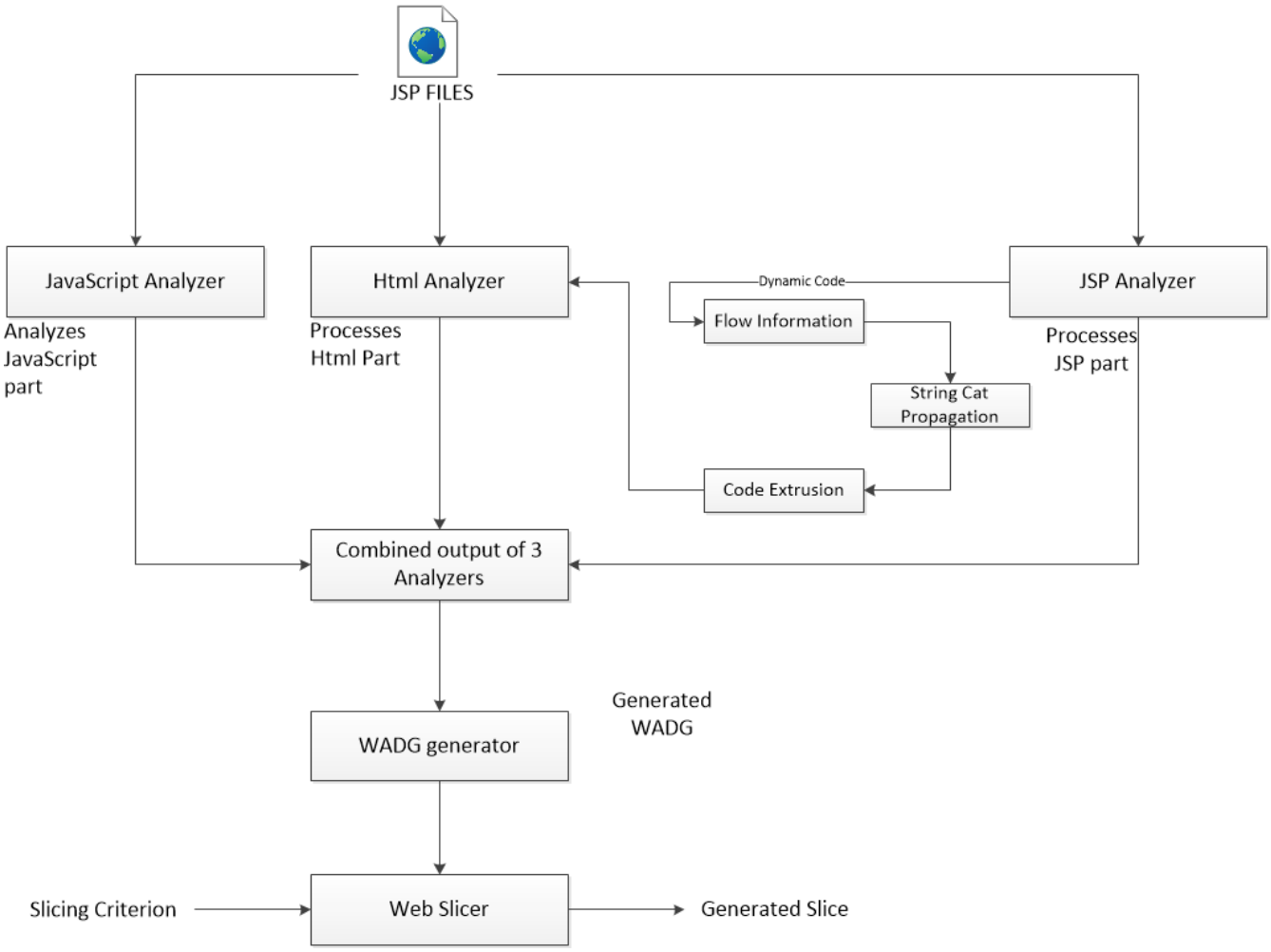

Figure 6.1: A systematic block diagram for WebSlicer

file and also provides different dependencies by different notations. It generates the WADG completely and it gives its output to the WebSlicer. The WebSlicer receives the generated WADG file and the slicing criterion as input. Then, it process the WADG file and finds the node given by the slicing criterion. Finally, it generates the sliced WADG by applying the slicing algorithm on the input WADG and writes it to a .gv file in which the sliced nodes are marked.

6.2 Case Study

In-order to verify the ability of our designed WebSlicer tool, we have preformed three case studies. In each case study, we have taken one JSP program, then generated it's WADG. After that we have computed slices for the same program and WADG, by taking different slicing criteria. Choosing of slicing criteria is a difficult task, but we have fixed the output statements, like print statement or method return statements, as slicing criteria. The details of case studies is given in Table-6.1.

Table 6.1: Case study of various applications

Sl. No.	**Name of Application**	**Details**
1	Calculator	Performs mathematical operations
2	Book Management System	Storage and issue of text books for an institute
3	Java EE Training	Tutorials and practice questions of Java

6.3 Findings

First we have generated the WADG for each JSP applications, and the details are given in Table-6.2. Then we have computed several slices by supplying different slicing criteria. To summarize our finding, we have calculated the average slice size, which is the addition of individual slice size and number of slices computed. Similarly, we have computed the average slicing time, which is required to compute slices. Table-6.3 consists of all these findings.

Table 6.2: Details of generated WADG for case studies

Name of Application	**LOC**	**No Of nodes in WADG**	**No Of Edges in WADG**	**Time to Generate WADG**
Calculator	250	130	149	31.07 ms
Book Management System	834	527	612	114.39 ms
Java EE Training	435	281	288	54.57 ms

Table 6.3: Outcome of the case studies

Sl. No.	**Name of Application**	**Average Slice Size**	**Average Slice Time**
1	Calculator	7	2.61 ms
2	Book Management System	24	5.83 ms
3	Java EE Training	18	3.17 ms

6.4 Summary

In this chapter, the overall implementation of the work was discussed by first describing the experimental setup used for performing various case studies. A tabular representation of the case studies along with their finding is shown. The overall model of the setup is shown with the help of a flow diagram. The flow diagram shows the various modules of WADG generation and slicing.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

In this thesis, we have proposed a technique for slicing of JSP programs. Most of the existing techniques are based on byte-code analysis of the program, which causes more complex SDGs to represent and design. In our approach, we have used the source-code based program analysis for construction of the WADG for a given JSP program. We have applied a two-phase slicing algorithm to compute slices. Then to verify our developed tool, we have conducted some case studies. The slices computed by our tool is check manually to be correct. We have found that all the slices computed by our technique is precise and correct.

7.2 Future Work

The technique can be extended for slicing of web applications in PHP and other languages by just redoing the syntax analysis. Moreover,this work can be further enhanced and used for regression testing of web applications.

Bibliography

[1] W. Tsai, Xiaoying Bai, Ray Paul, and Lian Yu. Scenario-based functional regression testing. In *Computer Software and Applications Conference, 2001. COMPSAC 2001. 25th Annual International*, pages 496–501. IEEE, 2001.

[2] Wes Masri, Andy Podgurski, and David Leon. Detecting and debugging insecure information flows. In *Software Reliability Engineering, 2004. ISSRE 2004. 15th International Symposium on*, pages 198–209. IEEE, 2004.

[3] Lei Xu, Baowen Xu, Zhenqiang Chen, Jixiang Jiang, and Huowang Chen. Regression testing for web applications based on slicing. In *Computer Software and Applications Conference, 2003. COMPSAC 2003. Proceedings. 27th Annual International*, pages 652–656. IEEE, 2003.

[4] Junhua Wu, Baowen Xu, and Jixiang Jiang. Slicing web application based on hyper graph. In *Cyberworlds, 2004 International Conference on*, pages 177–181. IEEE, 2004.

[5] Mark Weiser. Program slicing. In *Proceedings of the 5th international conference on Software engineering*, pages 439–449. IEEE Press, 1981.

[6] Mark Weiser. Program slicing. In *Proceedings of the 5th international conference on Software engineering*, pages 439–449. IEEE Press, 1981.

[7] Filippo Ricca and Paolo Tonella. Web application slicing. In *Proceedings of the IEEE International Conference on Software Maintenance (ICSM'01)*, page 148. IEEE Computer Society, 2001.

[8] Chengying Mao. Slicing web service-based software. In *Service-Oriented Computing and Applications (SOCA), 2009 IEEE International Conference on*, pages 1–8. IEEE, 2009.

[9] David W Binkley and Keith Brian Gallagher. Program slicing. *Advances in Computers*, 43:1–50, 1996.

[10] Andrea De Lucia. Program slicing: Methods and applications. In *2013 IEEE 13th International Working Conference on Source Code Analysis and Manipulation (SCAM)*, pages 0144–0144. IEEE Computer Society, 2001.

[11] Paolo Tonella and Filippo Ricca. Web application slicing in presence of dynamic code generation. *Automated Software Engineering*, 12(2):259–288, 2005.

[12] Susan Horwitz, Thomas Reps, and David Binkley. Interprocedural slicing using dependence graphs. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 12(1):26–60, 1990.

[13] Filippo Ricca and Paolo Tonella. Construction of the system dependence graph for web application slicing. In *Source Code Analysis and Manipulation, 2002. Proceedings. Second IEEE International Workshop on*, pages 123–132. IEEE, 2002.

[14] Madhusmita Sahu and Durga Prasad Mohapatra. Slicing java server pages application. *IEEE International Conference on Inofrmation Technology, ICIT*, 2008.

[15] H Casalánguida and JE Durán. Aspect oriented navigation modeling for web applications based on uml. *Latin America Transactions, IEEE (Revista IEEE America Latina)*, 7(1):92–100, 2009.

[16] Josip Maras, Jan Carlson, and Ivica Crnkovic. Client-side web application slicing. In *Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering*, pages 504–507. IEEE Computer Society, 2011.